# Einführung in *Mathematica*
# -- Teil II --

zur Vorlesung *Mathematische Methoden der Physik* im WiSe 2014/15

: : Prof. Dr. Norbert Dragon und PD Dr. Michael Flohr : : 14. 11. 2014 : :

---

## Analysis

*Mathematica* kann exellent mit allem umgehen, was mit Analysis zu tun hat. Differenzieren haben wir bereits in Teil I kennen gelernt. Natürlich kann *Mathematica* auch integrieren, und zwar so gut, dass, wenn es denn überhaupt eine in elementaren Funktionen ausdrückbare Stammfunktion gibt, sie von *Mathematica* auch gefunden wird. Und *Mathematica* löst auch Differentialgleichungen - die fundamentalen Gleichungen in der Physik.

### Differentialgleichungen

Analytische Integration (Man beachte die Integrationskonstante `C[1]`).

```
DSolve[y'[x] - x y[x] == 0, y[x], x]
```

$$\left\{\left\{y[x] \to e^{\frac{x^2}{2}} C[1]\right\}\right\}$$

Beim Zuweisen zu einer neuen Funktion `z` ist wiederum der Ersetzungsoperator notwendig; `[[1]]` "löst" sozusagen die verschachtelnde Liste auf.

```
z[x_] = y[x] /. DSolve[y'[x] - x y[x] == 0, y[x], x][[1]]
```

$$e^{\frac{x^2}{2}} C[1]$$

Nun kann, bis auf die verbleibende Konstante, mit der neuen Funktion numerisch gerechnet werden ...

```
z[1]
```

$$\sqrt{e}\, C[1]$$

```
N[%]
```

```
1.64872 C[1]
```

Die Eleminierung der Konstanten erfolgt, wie gehabt, über eine Ersetzung mit einer Transformation-sregel (`I` ist die komplexe Zahl, siehe Teil I):

```
% /. C[1] → I
```

```
0. + 1.64872 i
```

Analytische Integration mit Randbedingung:

```
DSolve[{y''[x] + ω^2 y[x] == 0, y[0] == 1, y'[0] == 1 / 2}, y[x], x]
```

$$\left\{\left\{y[x] \to \frac{2\,\omega\,\text{Cos}[x\,\omega] + \text{Sin}[x\,\omega]}{2\,\omega}\right\}\right\}$$

Allgemeine Lösung einer Differentialgleichung in Form der sog. “reinen” Funktion (beachte, dass das Argument bei `y` nicht mit angegeben wird):

```
DSolve[y''[x] + ω^2 y[x] == 0, y, x]
```

```
{{y → Function[{x}, C[1] Cos[x ω] + C[2] Sin[x ω]]}}
```

Lösung für einen bestimmten Fall durch Substitution, vergleiche mit dem ersten Beispiel:

```
(y[x] /. %)[[1]]
```

```
C[1] Cos[x ω] + C[2] Sin[x ω]
```

*Hinweis*: Die Substitutionsanweisung schreibt man vorteilhafterweise in runde Klammern, bevor die Liste aufgelöst wird, denn *Mathematica* ist sehr logisch: Geschweifte Klammern erzeugten gleich wieder eine neue Liste, für die eine weitere Operation mit `[[1]]` nötig wäre.

```
{y[x] /. %%}[[1]][[1]]
```

```
C[1] Cos[x ω] + C[2] Sin[x ω]
```

## Vektoranalysis

Die Vorlesung “Mathematische Methoden der Physik” macht Sie vor allem mit den Techniken der Vektoranalysis vertraut, die *Mathematica* natürlich auch beherrscht.

```
Clear[a, b, r, x, y, z, f, g, h]
```

Früher musste man hier erst ein Zusatzpaket laden, das aber mittlerweile in den Kern der grundlegenden *Mathematica*i Funktionen, die immer zur Verfügung stehen, integriert wurde. Überhaupt gibt es so viele spezielle Funktionen und Algorithmen in *Mathematica,* dass es keinen Sinn macht, alle diese bei jedem Programmstart vollständig zu laden. Statt dessen wird nur der als absolut essentiell erachtete Teil immer geladen, den Rest kann man bei Bedarf in Form von Paketen dazuladen, die sich jeweiligen speziellen Themen widmen.

```
(* <<VectorAnalysis`*)
```

*Hinweis*: Früher musste man zuerst das Koordinatensystems und seine Variablen festlegen (*Mathematica* gibt zwar `Cartesian`, d.h. “kartesisch”, aber konsistent zu seinen Groß- und Kleinschreibungsregeln `Xx`,`Yy` und `Zz` vor. Ein anderes mögliches Koordinatensystem ist z. B. `Spherical` mit den Variablen `Rr`, `Ttheta`, `Pphi`). In der jetzigen Implementation wird stärker betont, dass die Stukturen im Prinzip unabhängig vom konkreten Koordinatensystem sind.

```
(* SetCoordinates[Cartesian[x,y,z]] *)
```

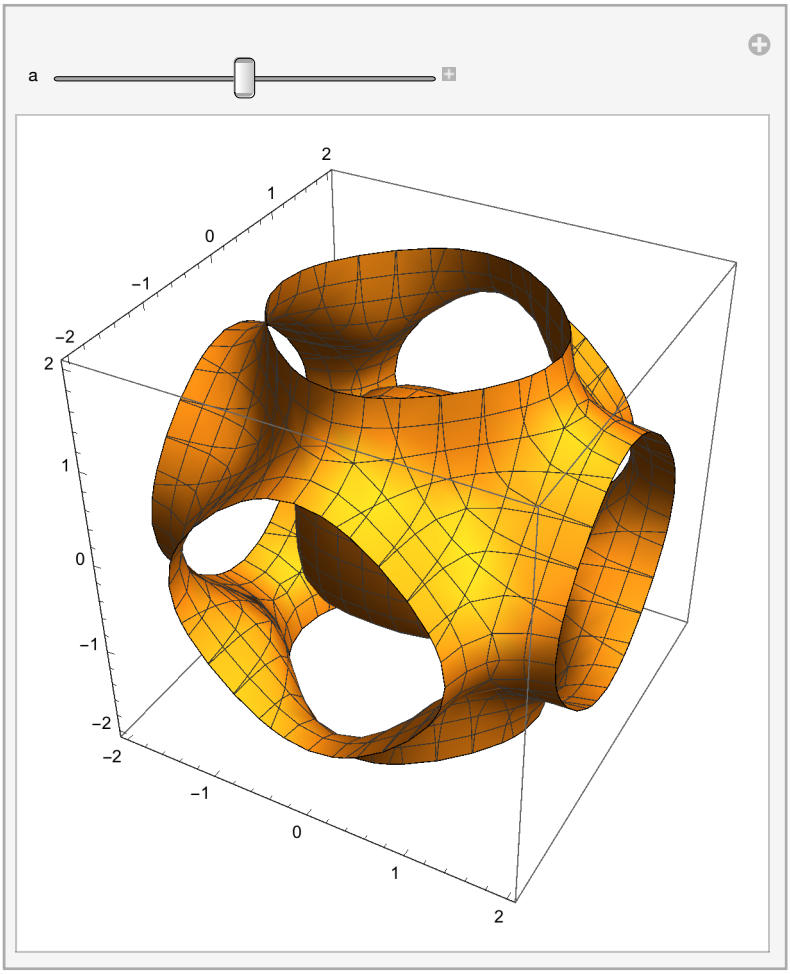
Definition einer (skalaren) Funktion, als zum Beispiel eines Potentials eines konservativen Kraftfeldes:

In[232]:=
```
f[x_, y_, z_] := x^2 (3 - 2 y^2) - 2 (x^2 + y^2) z^2 + 3 (y^2 + z^2)
```

Von besonderem Interesse sind die Punkte, an denen das Potential den gleichen Wert hat. Hier plotten wir so eine Äquipotentialfläche:

In[233]:=
```
Manipulate[ContourPlot3D[f[x, y, z] == a, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}],
 {{a, 3}, 1, 5}, SaveDefinitions → True]
```

Out[233]=




Haben wir ein Potential, ist das Kraftfeld durch den negativen Gradienten des Potentials gegeben. Berechnung des Gradienten von **f** im Cartesischen Koordinatensystem:

```
g = Grad[f[x, y, z], {x, y, z}, "Cartesian"]
```

$$\left\{2\,x\left(3-2\,y^2\right)-4\,x\,z^2,\ 6\,y-4\,x^2\,y-4\,y\,z^2,\ 6\,z-4\left(x^2+y^2\right)z\right\}$$

*Achtung*: **g** ist keine Funktion gemäß der für **f** gebrauchten Definition, sondern eine “vektorartige” Liste!

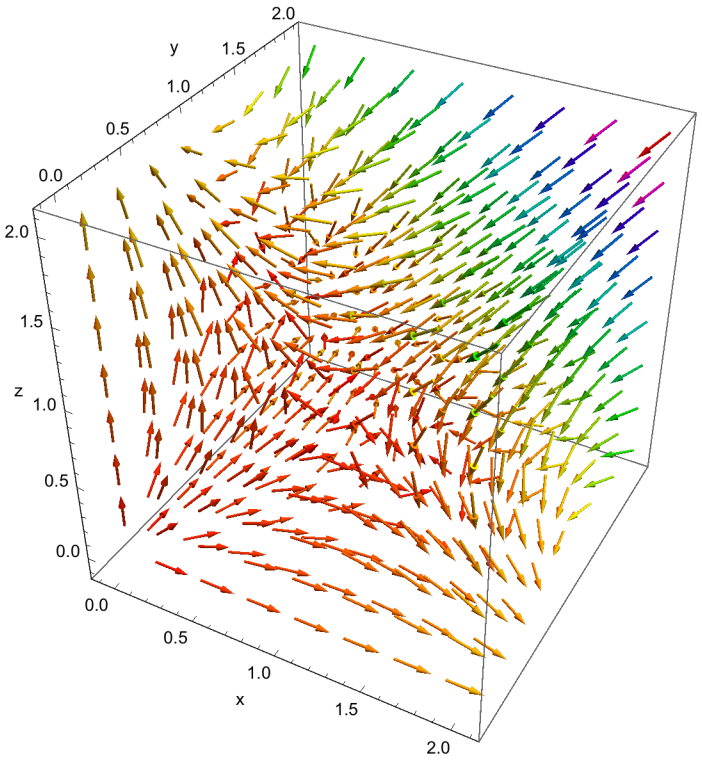
Wenn wir also mal einen Gradienten in Kugelkoordinaten brauchen ...

```
Grad[Φ[r, θ, ϕ], {r, θ, ϕ}, "Spherical"]
```

$$\left\{\Phi^{(1,0,0)}[r,\theta,\phi],\ \frac{\Phi^{(0,1,0)}[r,\theta,\phi]}{r},\ \frac{\text{Csc}[\theta]\,\Phi^{(0,0,1)}[r,\theta,\phi]}{r}\right\}$$

Darstellung eines dreidimensionalen Vektorfeldes: Unter den benutzten Darstellungsoptionen sei die universelle, auch in **Plot** verwendbare **AxesLabel** hervorgehoben.

```
VectorPlot3D[g, {x, 0, 2}, {y, 0, 2}, {z, 0, 2},
 AxesLabel → {"x", "y", "z"}, VectorStyle → "Arrow3D",
 VectorScale → {Small, Small, None}, VectorColorFunction → Hue]
```




Wir berechnen nun die zwei grundlegenden Differentationen für Vektorfelder, die Divergenz und die Rotation. Letztere sollte für ein Gradientenfeld natürlich verschwinden:

```
h = Div[g, {x, y, z}, "Cartesian"]
r = Curl[g, {x, y, z}, "Cartesian"]
```

$$12 - 4\,x^2 + 2\left(3 - 2\,y^2\right) - 4\left(x^2 + y^2\right) - 8\,z^2$$

$$\{0, 0, 0\}$$

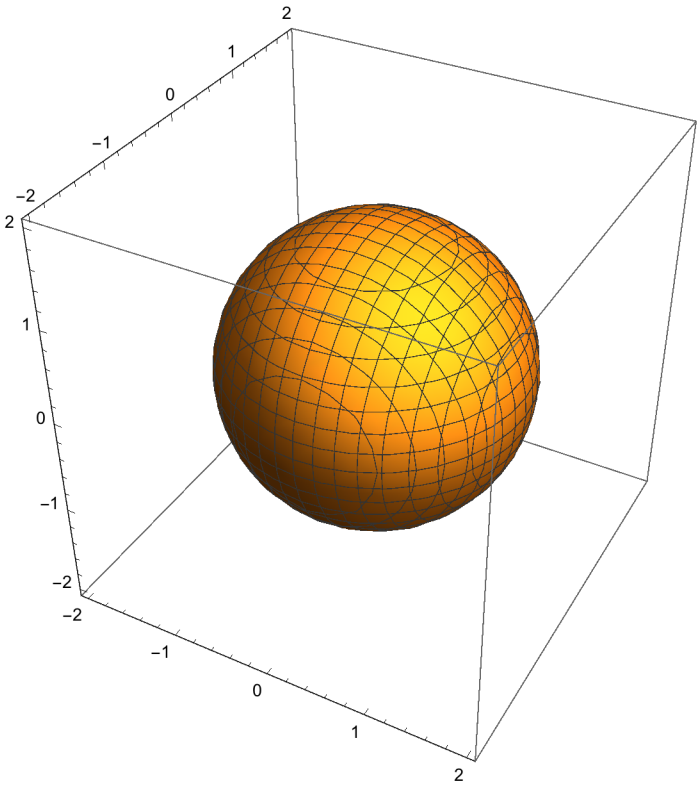
```
FullSimplify[%%]
```

$$-2\left(-9 + 4\,x^2 + 4\,y^2 + 4\,z^2\right)$$

*Achtung*: **h** ist ebenfalls keine Funktion gemäß der für **f** gebrauchten Definition, sondern eine “skalarartige” (also einelementige) Liste!

Wie man nach der Vereinfachung mittels **FullSimplify** sieht, ist diese Funktion rotationssymmetrisch:

```
ContourPlot3D[h == 0, {x, -2, 2}, {y, -2, 2}, {z, -2, 2}]
```




# Listen

*Mathematica* arbeitet intern unglaublich stark mit dem Konzept von Listen. Das kann man daher ausnutzen, um sehr effizienten Code zu schreiben, oder manche mathematischen Probleme sehr einfach in *Mathematica* zu implementieren. Deshalb ist es wichtig, die grundlegenden Befehle zum Hantieren mit Listen zu kennen.

## Listenerzeugung

Ein einfaches Beispiel zum Einstieg:

```
Table[n, {n, 0, 16}]
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}
```

```
Table[n, {n, 0, 16, 2}]
```

```
{0, 2, 4, 6, 8, 10, 12, 14, 16}
```

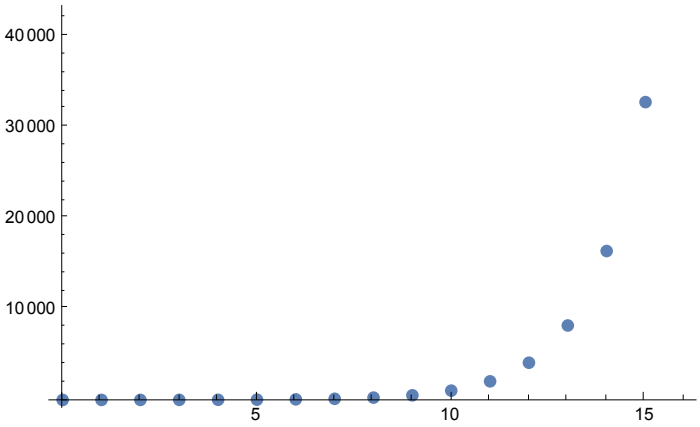
Tabellen sind im wesentlichen geordnete Listen. Und sie können ganz einfach, ineinander verschachtelt werden. Wir kennen das im Grunde schon ähnlich von den **Array**, die wir in Teil I zur Erzeugung von Vektoren und Matrizen genutzt haben. **Table** kann also auch eine Liste von Listen generieren, beispielsweise Wertetabellen.

```
Table[{n, 2^n}, {n, 0, 16}]
```

```
{{0, 1}, {1, 2}, {2, 4}, {3, 8}, {4, 16}, {5, 32},
 {6, 64}, {7, 128}, {8, 256}, {9, 512}, {10, 1024}, {11, 2048},
 {12, 4096}, {13, 8192}, {14, 16 384}, {15, 32 768}, {16, 65 536}}
```

Listen dieser Art mit numerischen Daten lassen sich unkompliziert graphisch darstellen.

```
ListPlot[%]
```




Symolisch geht's mit der Listenerzeugung freilich auch:

```
Table[(a + b)^n, {n, 0, 4}]
```

$$\{1, a+b, (a+b)^2, (a+b)^3, (a+b)^4\}$$

```
liste = Expand[%]
```

$$\{1, a+b, a^2+2\,a\,b+b^2, a^3+3\,a^2\,b+3\,a\,b^2+b^3, a^4+4\,a^3\,b+6\,a^2\,b^2+4\,a\,b^3+b^4\}$$

## Listenmanipulationen

Zunächst wird beispielhaft das erste Listenelement der vorletzten Programmausgabe entfernt ...

```
Drop[%%, 1]
```

$$\{a+b, (a+b)^2, (a+b)^3, (a+b)^4\}$$

... und dann das erste "extrahiert".

```
First[%]
```

$$a+b$$

Den letzten Listeneintrag erhält man "natürlich" mit **Last**!

```
Last[liste]
```

$$a^4+4\,a^3\,b+6\,a^2\,b^2+4\,a\,b^3+b^4$$

Ausschneiden von Listenteilen erlaubt **Part**.

```
Part[liste, 2 ;; 3]
```

$$\{a+b, a^2+2\,a\,b+b^2\}$$

Oder alternativ auf diesem Weg (vergleiche mit dem Extrahieren von **Array** Elementen in Teil I):

```
liste[[2 ;; 3]]
```

$$\{a+b, a^2+2\,a\,b+b^2\}$$

Der Zugriff auf ein einzelnes Element erfolgt in gewohnter Manier:

```
%[[1]]
```

$$a+b$$

```
Table[{n, 1/n}, {n, 1, 7}]
```

$$\left\{\{1, 1\}, \left\{2, \frac{1}{2}\right\}, \left\{3, \frac{1}{3}\right\}, \left\{4, \frac{1}{4}\right\}, \left\{5, \frac{1}{5}\right\}, \left\{6, \frac{1}{6}\right\}, \left\{7, \frac{1}{7}\right\}\right\}$$

Ein extrem nützlicher Befehl: **Flatten** reduziert die beiden Ebenen zuvor generierter Liste auf eine.

```
Flatten[%]
```

$$\left\{1, 1, 2, \frac{1}{2}, 3, \frac{1}{3}, 4, \frac{1}{4}, 5, \frac{1}{5}, 6, \frac{1}{6}, 7, \frac{1}{7}\right\}$$

Die Summe aller Listenelemente läßt sich mit **Total** berechnen.

```
Total[%]
```

$$\frac{4283}{140}$$

# Bedingungen

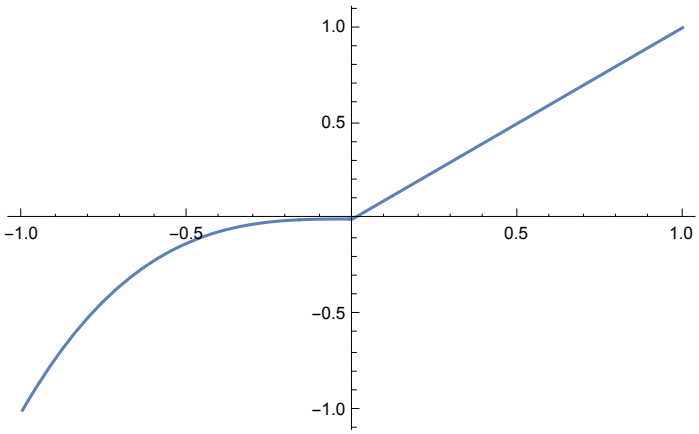
*Mathematica* ist nicht nur eine Ansammlung zahlloser Befehle und Datenstrukturen zum Umgang mit mathematischen Objekten, sondern eine vollwertige “High-Level” Programmiersprache. Wie jede vernünftige Programmiersprache bietet es die Möglichkeit, Bedingungen zu formulieren, auszuwerten, und dann in Abhängigkeit vom Ergebnis unterschiedlich weiter zu verfahren.

## Abschnittsweise definierte Funktionen

Eine reelle Funktion, die auf zwei Intervallen unterschiedlich definiert ist,

```
f[x_] := If[x < 0, x^3, x]
```

```
Plot[f[x], {x, -1, 1}]
```




sowie deren bestimmtes Integral,

```
Integrate[f[x], {x, -1, 1}]
```

$$\frac{1}{4}$$

deren "Stammfunktion"

```
Integrate[f[x], x]
```

$$\begin{cases} \frac{x^4}{4} & x \leq 0 \\ \frac{x^2}{2} & \text{True} \end{cases}$$

und deren erste Ableitung

```
Clear[g]
```

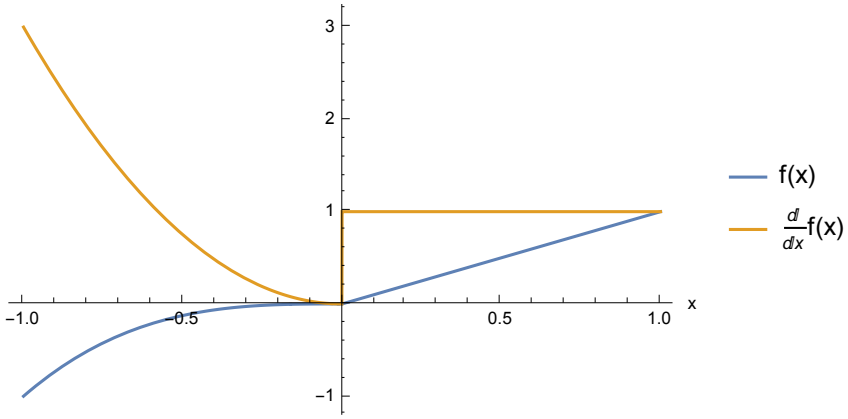
```
g[x_] = D[f[x], x]
```

```
If[x < 0, 3 x^2, 1]
```

Wir wollen uns das in einem Plot ansehen. Wir hätten bei diesem Plot allerdings auch gerne eine schöne Legende, die die verschiedenen Funktionen, die wir plotten, auflistet. Es gibt zahllose solche “Sonderwünsche”, für die es in *Mathematica* bereits Lösungen in Form von Zusatzpaketen gibt. Damit das Programm beim Starten jedoch nicht eine riesige Menge solcher Zusatzfunktionen laden muss, wird nur ein kleiner Teil grundlegender Funktionen geladen, der Rest kann in Form dieser Zusatzpakete, falls benötigt, jederzeit nachgeladen werden.
So musste das Paket **PlotLegends** zum Erzeugen einer Legende zuerst geladen werden. Das Laden eines Paketes geht mit folgender Syntax, wobei dieses spezielle Paket inzwischen zum grundlegenden Kern von *Mathematica* zählt und nicht mehr geladen werden müsste.

```
(* <<PlotLegends` *)
```

```
Plot[{f[x], g[x]}, {x, -1, 1}, AxesLabel → {"x"},
 PlotLegends → Placed[{"f(x)", "∂x f(x)"}, After]]
```




## Weitere Anwendungen

Hier kombinieren wir das bisher gelernte über Listen und Bedingungen in ein paar wenigen Beispielen, die sich um Primzahlen drehen.

```
primzahlen = Table[Prime[i], {i, 20}]
```

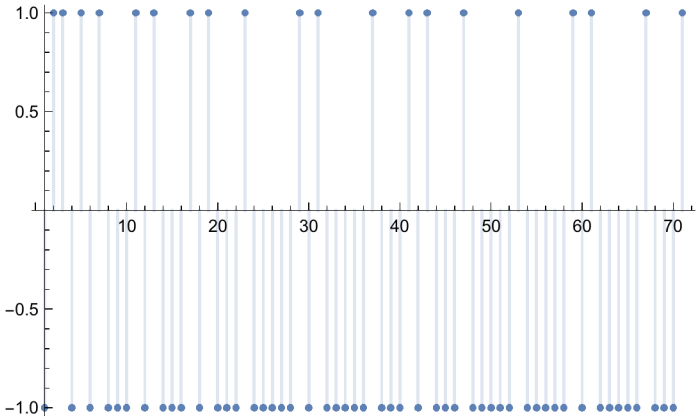
```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71}
```

```
j = Last[primzahlen]
```

```
71
```

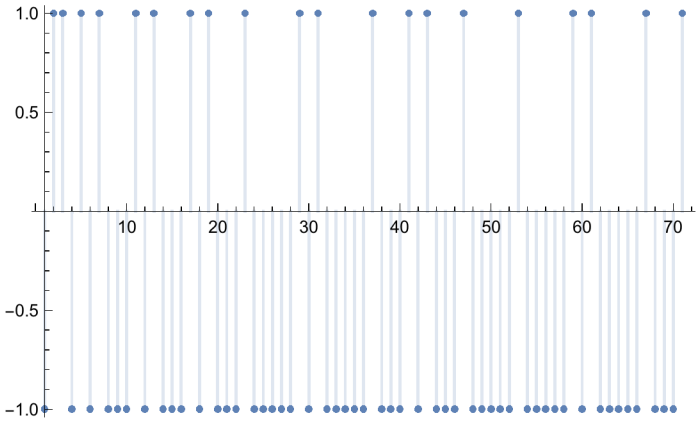
Die Funktion **MemberQ** durchsucht eine Liste darauf hin, ob ein bestimmtes Element enthalten ist.

```
DiscretePlot[If[MemberQ[primzahlen, i], 1, -1], {i, 1, j}]
```




Durchsuchen ist aber eigentlich gar nicht nötig, da die Liste aufsteigend sortiert ist und aufsteigend "abgearbeitet" wird.

```
DiscretePlot[If[i == First[primzahlen], primzahlen = Drop[primzahlen, 1];
  1, -1], {i, 1, j}]
```




Achtung: Nach obiger "Abarbeitung" der Primzahlenliste ist diese leer, denn die Liste wurde tatsächlich während der Auswertung bei jedem Schritt um ein Element verkürzt.

```
primzahlen
```

```
{}
```

# Logische Ausdrücke

*Mathematica* kann in vielfältiger Weise logische Ausdrücke konstruieren und auswerten. Jeder auswertbare logische Ausdruck basiert auf Grundeinheiten, die meist als Vergleich formuliert werden können. Alle Vergleiche können nur zwei Werte annehmen: Wahr oder falsch.

```
2^0 == 1
```

```
True
```

```
2 < 0
```

```
False
```

### Verknüpfung zweier Vergleiche mit "und" bzw. "oder"

```
3 > E && 3 < Pi
```

```
True
```

Äquivalente Eingabe:

```
3 > E ⋀ 3 < Pi
```

```
True
```

```
3 < E || 3 < Pi
```

```
True
```

Äquivalente Eingabe:

```
3 > E ⋁ 3 > Pi
```

```
True
```

### Negation

```
! (a > b)
```

```
a ≤ b
```

### Verschachtelte `If`-Befehle

```
Integrate[If[x < 0, If[x == -1, 1, 2], 3], x]
```

$$\begin{cases} 2x & x \leq -1 \,||\, -1 < x \leq 0 \\ 3x & \text{True} \end{cases}$$

## Schleifenstrukturen und Iteration

Der nächste wichtige Typ von Strukturbefehlen, der *Mathematica* zu einer vollen und extrem leistungsfähigen Programmiersprache macht, sind Schleifen, mit denen die gleiche Sequenz von Befehlen mehrfach ausgeführt werdden können.

```
i = 0;
While[i < 3, Print[i]; i = i + 1]
```

```
0
```

```
1
```

```
2
```

Eine klassische "`for`"-Schleife, wie sie wohl nahezu jede (imperative) Programmiersprache kennt:

```
For[i = 0, i < 3, i = i + 1, Print[i]]
```

```
0
```

```
1
```

```
2
```

In *Mathematica* gibt es auch das einfachere, zu **`For`** äquivalente **`Do`**,

```
Do[Print[i], {i, 0, 2}]
```

```
0
1
2
```

welches von der Struktur beispielsweise analog zu **Table** ist.

```
Table[i, {i, 0, 2}]
```

```
{0, 1, 2}
```

## Anwendungen

Wozu ist das gut? Hier ein paar wirklich extrem simple Beispiele. Wie alles in *Mathematica* können auch diese Strukturen beliebig ineinander verschachtelt werden. Es hat auch seinen Grund, dass viele Dinge in *Mathematica* auf verschiedene Weisen erreicht werden können. So hat jede der verschiedenen Methoden einer **For** Schleife besondere Vorzüge für bestimmte Anwendungen, was zu schnellerem und effizienterem Code führt, wenn man sich damit auskennt.

```
For[i = 0, i < 3, i = i + 1, k = i]
```

```
k
```

```
2
```

Eine (Summations)schleife in mathematisch etwas vertrauterer Darstellung

```
k = Sum[1, {i, 1, 10}]
```

```
10
```

Dieselbe sieht "programmiert" wie folgt aus:

```
k = 0;
For[i = 1, i ≤ 10, i = i + 1, k = k + 1]
k
```

```
10
```

Ein weiteres Beispiel, die Gauß-Summe

```
k = ∑_{i=1}^{10} i
```

```
55
```

```
k = 0;
For[i = 1, i ≤ 10, i = i + 1, k = k + i]
k
```

```
55
```

Am Rande sei erwähnt, dass *Mathamtica* die Gauß-Summe natürlich auch für eine unbestimmte obere Grenze symbolisch ausrechnen kann.

```
k = ∑_{i=1}^{N} i
```

$$\frac{1}{2} N (1 + N)$$

### Fakultät, iterativ

Und hier noch ein Klassiker, die iterative Definition der Fakultät:

```
fi[n_] := Product[m, {m, 2, n}]
```

```
fi[137]
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 ↓
  972 746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 ↓
  314 826 377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 ↓
  000 000 000 000 000 000 000 000 000
```

Und zur Kontrolle hier mit dem in *Mathematica* eingebauten Befehl

```
137 !
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 ↓
  972 746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 ↓
  314 826 377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 ↓
  000 000 000 000 000 000 000 000 000
```

## Rekursion

*Mathematica* kann hervorragend mit rekursiv definierten Strukturen umgehen. Das ist zwar nicht immer die effizientere Lösung, aber oft elegant und in vielen Fällen einfacher zu programmieren

### Fakultät, rekursiv

```
fr[n_] := n fr[n - 1]
```

```
fr[1] = 1;
```

```
fr[137]
```

```
5 012 888 748 274 991 661 034 926 292 112 253 883 237 205 694 398 754 483 388 962 668 892 510 ↓
  972 746 226 260 034 675 717 797 072 343 372 830 591 567 227 826 571 884 373 881 355 612 819 ↓
  314 826 377 917 827 129 740 056 802 397 016 509 378 163 883 274 055 583 382 110 208 000 000 ↓
  000 000 000 000 000 000 000 000 000
```

Die Funktion **Timing** gibt die Verarbeitungszeit in Sekunden an.

```
Timing[fr[10 000];]
```

```
{0.001486, Null}
```

Um eine Systemvariable wie **$RecursionLimit** nur in einer Eingabezeile zu verändern, muß die beeinflußte Funktion in **Block** verschachtelt werden.

```
Timing[Block[{$RecursionLimit = 10 004}, rf[10 000];]]
```

```
{0.000010, Null}
```

Hier ein Geschwindigkeitsvergleich der drei Möglichkeiten:

```
Timing[Block[{$RecursionLimit = 10 004}, rf[10 000];]]
```

$$\{9. \times 10^{-6}, \text{Null}\}$$

```
Timing[fi[10 000];]
```

```
{0.004359, Null}
```

```
Timing[10 000!;]
```

```
{0.000460, Null}
```

Aha! Die iterative Definition ist die schlechteste. Dies liegt daran, dass bei der iterativen Definition sehr viele Zwischenergebnisse erzeugt werden, was wiederum daran liegt, dass die **Product** Funktion im Grunde in einer Art **For** Schleife implementiert ist. Die rekursive Definition hingegen arbeitet “in place”, und erledigt daher die 10.000-fache Multiplikation effizienter. Und die eingebaute Funktion ist ohnehin wesentlich komplizierter programmiert, und hat einen sogenannten “overhead”, weil sie erst einmal prüft, ob das Argument eine einfache Berechnung mit natürlichen Zahlen zulässt, eine symbolisch exakte Berechnung, eine numerische, oder gar keine.
Ergebnis des Geschwindigkeitsvergleiches. Die eingebauten Funktionen von *Mathematica* sind hochoptimiert und meist unschlagbar schnell. Dass die rekursive Definition hier gewinnt, liegt daran, dass *Mathematica* einmal berechnete Werte cached, und das die eingebaute Funktion für die Fakultät wesentlich mehr kann. Dahinter steckt nämlich die Γ-Funktion, die analytische Fortsetzung der Fakultät auf die komplexen Zahlen.

Und wieder eine Randbemerkung: Die interne Definition der Fakultät ist in der Tat in Wirklichkeit viel allgemeiner, denn sie kann auch auf beliebige reelle (und komplexe!) Zahlen angewandt werden. In Wirklichkeit ist dies die Gammafunktion:

```
5.5!
```

```
287.885
```

$$\frac{11}{2}!$$

$$\frac{10\,395\,\sqrt{\pi}}{64}$$